Quality Software

ARK Systems USA
P.O. Box 23
Santa Clara, CA 95052-0023

OS-9/680x0 Program

IBF

IEEE488/GP-IB File Manager

A Brief Look

Rev. A 2/92

Printed in U.S.A.

We express special thanks to Mr. Steven Weller of Windsor Systems and Mr. Toshikuni Osogoe of Osque Systems for their contributions to the improvement of this booklet.

Printing History

Rev. A 2/92 First printing.

Table of Contents

Chapter 1
Introduction

IBF is a file manager for the OS-9/68000 real-time operating system. IBF supports IEEE488, an industry standard interface bus primarily for test and measurement applications. IBF lets your application programs access devices on an IEEE488 bus just like other OS-9 file devices. This provides simplicity and uniformity of device access to your application programs.

IBF was developed in 1988 by an ex-HP senior engineer at ARK Corporation in Japan to provide a powerful software base to the interface between OS-9 and IEEE488. Since then, IBF has been ported to a number of different platforms and has proved the correctness of the original design concept.

Although this booklet was designed primarily for providing a general introduction to IBF to those who are looking for powerful and elegant software support for IEEE488 under OS-9/68000, those customers who have already been using IBF may take advantage of the topics and tips described herein.

If you need additional copies of this booklet, send check or money order to ARK Systems USA, P.O. Box 23, Santa Clara, CA 95052-0023. 1 copy: $1.00; 10 copies: $8.00; 50 copies: $30.00. Foreign countries other than Canada and Mexico add 50%. Remittance must be made in U.S. funds.

Chapter 2
IBF Overview

2.1 Modular Software Architecture

IBF, like other OS-9 file managers, is aimed at providing a logically unified means of accessing its devices.

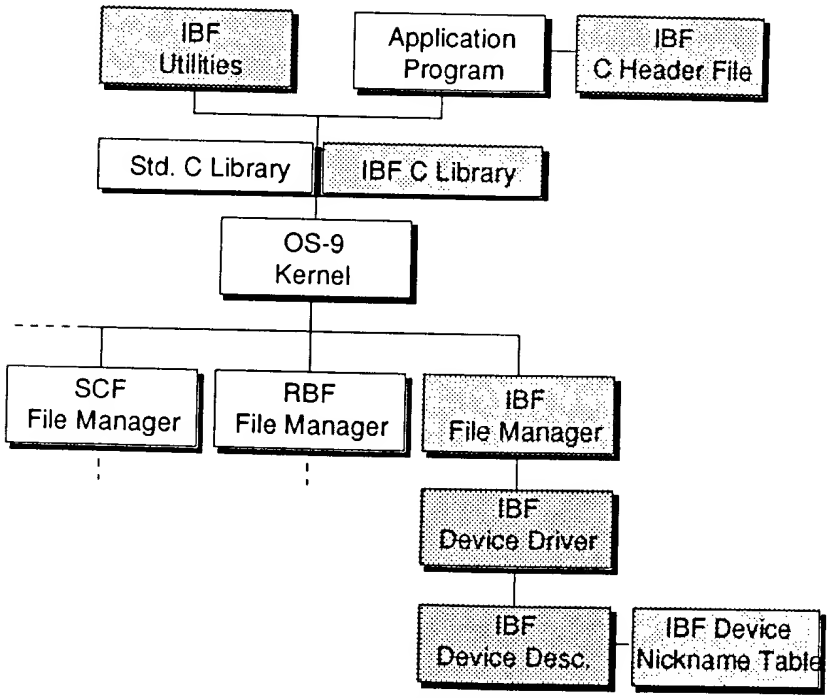

Fig. 2.1 IBF Software Architecture.

As shown in the module diagram in the previous page, IBF is no different from other OS-9 file managers such as SCF and RBF, though a few components have been added to provide more power to IBF.

Most IBF functions are implemented as system calls: the standard I$Read/I$Write/I$ReadLn/I$WritLn functions work straightforwardly the same as other OS-9 file managers such as RBF and SCF; IEEE488-specific functions such as serial and parallel polls and other bus management functions are implemented as I$GetStt and I$SetStt calls.

2.2 Easy Programming

IBF's implementation as a file manager allows you to use most standard C I/O library functions such as **read()**, **write()**, and even your favorite **printf()** for IBF devices. That's not all: several standard OS-9 utility programs such as *ECHO* and *LIST* are usable as well. For example, "`list file>/ib0/1`" will list your file on an IEEE488 printer at address 1.

For IBF-specific operations, a set of C library functions are bundled as an interface to the file manager. Some of the library functions are compatible with Hewlett-Packard's HP-UX (UNIX) DIO library functions. (Hewlett-Packard invented the IEEE488 bus.) See the **Specification** chapter for a list of IBF library functions.

Refer to a portion of a simplified example C program in the next page that performs a single measurement on a digital multimeter:


```
FILE *dvm;
char buf[80];
  fp = fopen("/ib0/dvm_hp3478a", "r+");
  fputs("TE\n", fp);
  fgets(buf, sizeof buf, fp);
  printf("Voltage = %s", buf);
```

2.3 Device Classification

One of IBF's elegant features is its *device classification*. IBF classifies its paths into two classes: an entire bus device is considered to represent the raw interface to the bus; an auto-addressed device represents an individual device (other than oneself) on the bus.

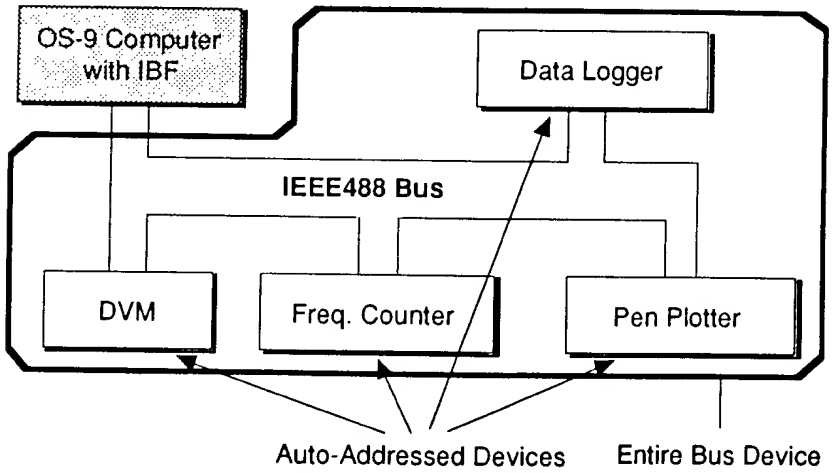

Fig. 2.2 Entire Bus Device and Auto-Addressed Devices.

This classification concept has made it possible to implement both complex bus management operations and basic data transfer

operations with ease.

For example, an entire bus device path name "/ib0" is used to send a trigger to all the devices on the bus and an auto-addressed device path name "/ib0/dvm1" is used to read data from a DVM (digital multimeter) named "dvm1" on the same bus as "/ib0." Use of these different classes of devices may be intermixed by an application program.

2.4 Fast Block Transfer

IBF performs "block-oriented" transfers: the file manager and the device driver transfer data to each other by the block, rather than by the character as SCF does. The data bytes are usually transferred directly between the user's buffer and the LSI.

IBF also features synchronized operations: thanks to IEEE488's three-wire handshaking, data transfers are not initiated until a process explicitly issues either a read or write call. No "intermediate data buffer" to store unread or unwritten data bytes. This ensures integrity of data from different devices on the bus.

2.5 Portability and Compatibility

Since IBF is not a hardware-dependent library set as you may find in other implementations but a *file manager* with support programs, application programs using IBF are portable among different platforms without re-compilation. The device nickname table that resolves a device name to its bus address even absorbs different bus address configurations.

To OEM users, IBF comes with a complete set of the example

device driver's source code files. IBF supports two popular IEEE-488 interface LSIs: NEC μPD7210 and TI TMS9914A. The example device driver also includes complete code for the MC68450 DMAC (Direct Memory Access Controller).

Currently, the IBF device drivers are written in assembly code. Don't worry; the porting work will be much easier than you may think. IBF device drivers were written with great portability and flexibility in mind; setting a few switch parameters in a definition file and reassembling the whole program will make the device driver fit most hardware platforms. Virtually no modification of the device driver's source code is necessary.

2.6 The Future

Although the development of the OS-9000 version of IBF is still underway as the time of printing, it will inherit all the power of and maintain a high level of compatibility with its predecessor, the OS-9/68000 version, while fully conforming to OS-9000's new features. Call one of our authorized distributors for availability information.

Chapter 3
Questions and Answers

This chapter provides answers to the questions we have often been asked. They will help you understand IBF more. If you have any question that is not found here, please feel free to write to us at P.O. Box 23, Santa Clara, CA 95052-0023 USA, or fax to 1-408-244-5358.

3.1 The IEEE488 Standard

Q: Which standard does IBF support?

A: IBF conforms to the IEEE488.1-1987 standard, the latest revision as of this booklet's publication date. Nevertheless, since the IEEE488 standard was first published in 1975, virtually nothing has changed. IEC625-1 and JIS1901 are practically the same as IEEE488.1.

Q: Does IBF support IEEE488.2?

A: Yes and no. IEEE488.2 defines higher level protocols such as common command syntax and data representation. Due to the nature of OS-9's file managers, IBF provides services at the level of IEEE488.1. Higher level protocols and services defined in IEEE488.2 should be provided by application programs or other service modules. Nevertheless, IBF provides satisfactory services to such higher level protocol programs.

Q: Which functions of IEEE488 does IBF support?

A: IBF is capable of all Talker/Listener/Controller functions except for few which are considered unnecessary to IBF's applications, such as talk/listen only. Nevertheless, the necessary functions may depend on the implementation (porting). For example, if the device using IBF has nothing to "trigger," the Device Trigger function can be omitted.

For a complete list of interface functions, refer to the next chapter titled "**SPECIFICATIONS.**"

Q: Can I apply IBF to an embedded controller controlled by another computer through an IEEE488 bus?

A: Yes. IBF is good for both the controller (master) device and the instrument (slave) devices. A controller device's implementation presumably has all Talker/Listener/Controller functions where that of an instrument device lacks the controller function.

3.2 Data Transfer

Q: Does IBF support DMA?

A: Yes. The example device driver bundled with an IBF PortPak includes code for the MC68450 DMAC (direct memory access controller). The use of DMA, however, is optional and it is up to the IBF licensee (implementer) to decide whether to use DMA. The example device drivers' DMA code can be easily enabled and disabled with an assembly switch option.

Q: What is IBF's maximum data transfer rate?

A: IBF is a so-called block-oriented file manager: it transfers data

by the block, rather than by the character as SCF does. Therefore, it performs much faster than SCF devices even without DMA.

The actual transfer rate depends on many factors. According to the experimental reports of our own and our customers, a typical implementation will result in 3~20kB/s *without* DMA (all bytes are handled by interrupts) and 100~500kB/s *with* DMA.

Q: I thought IEEE488 defines a transfer rate of 1MB/s. Why is IBF so slow?
A: *IBF is not slow.* The IEEE488.1 defines the bus's *maximum* data transfer rate or capacity of 1MB/s. To achieve this, however, the following restrictive conditions are required (values in parenthesis are relaxed spec for a 250kB/s transfer rate): [1] the talker device uses a minimum multiline message settling time of 350ns (2μs); [2] the talk device uses 48mA three-state drivers (open-collector drivers); [3] the device capacitance on each lead except for REN and IFC is less than 50pF per device (100pF); [4] all devices in the system are powered on (2/3 of the devices); and [5] the interconnecting cable links are as short as 15m (20m) of total length per system with at least one equivalent load for each meter (2m) of cable. Besides, neither of popular IEEE488 LSIs (TMS9914A and μPD7210) will operate at 1MB/s.

Q: If an outside instrument device starts sending data when my application program using IBF is not yet ready to receive, are the bytes stored in some buffer so that my application can read them later, or are they lost?
A: They are neither stored nor lost. All IBF data transfers are performed "synchronously": unless your application program

explicitly issues an I$Read or I$ReadLn call and IBF is ready to receive data, no bytes are transferred. IEEE488's patented three-wire handshaking prevents data overrun and loss.

When your application program sends data bytes to an instrument device, on the other hand, the program will be blocked until all bytes have been transferred.

This characteristic of IBF presumably matches most IBF applications in which synchronization between the controller and instruments is important.

Q: I have a thermometer that sends out data only when there has been a change of more than 0.1°C. Do I have to keep polling the device?
A: No. IBF is capable of sending a signal to the application program (process) when a specific event occurs. The events include data ready, service request (SRQ) reception, and so forth. Your application may enter the sleep state so as not to waste CPU time or commit to doing other tasks.

Q: How does IBF treat record terminators?
A: IBF has two transfer modes: ***binary*** and ***text***, which respectively correspond to the I$Read/I$Write and I$ReadLn/I$WritLn system calls. In binary mode, there is no record terminator character: data is transferred "as is" without any modification until either EOI (End or Identify) is detected or the buffer gets full, whichever occurs first. Binary mode is useful when transferring large blocks of binary data such as scanner output.

In text mode, a user-definable EOS (End of Sequence) character

may terminate a data transfer even before EOI is detected or buffer becomes full. Either carriage return or line feed is usually chosen as EOS. IBF does the necessary conversion between different line terminators: OS-9's CR, UNIX-like LF, and MS-DOS-like CR+LF. Text mode is useful for transferring ASCII text strings and is similar to that of SCF (sequential file manager) but no other text editing such as backspace is provided.

3.3 Bus Management

Q: Do I have to learn how to address devices on the bus?

A: No. Among many of IBF's nice features, there is a concept called "auto-addressed device paths." Once a path is opened to an auto-addressed device, say "/ib0/17," IBF maintains a logical path to the device with the bus address of 17 as long as the path is open. The standard I$Read/ReadLn/Write/WritLn system calls and **read()/readln()/write()/writeln()** C library functions send address command preambles before transferring user data, so that it appears transparent to the application program.

Q: How can I perform IEEE488-specific operations?

A: IBF provides many IEEE488 specific services through I$GetStt and I$SetStt calls. Most of these services are also implemented as C library routines. Below are just a few of the functions found in a standard IBF program package:

`_ib_abort()` . . . Initializes devices' interface function.
`_ib_bus_status()` . . Inquires current hardware status.
`_ib_card_ppoll_resp()` Sets up my parallel poll response.
`_ib_clear()` . . . . . Sends device clear.
(Many more; see **SPECIFICATIONS** for a complete list.)

Q: My application has several processes running concurrently and accessing the same IEEE488 bus. Can I ensure the data integrity?

A: IBF has a function called "device lock." If a process issues a system call to lock a device, all other processes will get an E$DevBsy error when they access the device. The locked status is released when the process that locked the device issues an unlock system call or the process closes the path to the device. Of course, you can use OS-9's event semaphores to lock a device as well.

Q: Can I hook up more than one computer using IBF to the same bus?

A: Yes. You may put multiple computers using IBF on the same bus and transfer the controller's role back and forth between them. To implement such a multi-controller system, however, you need a fair amount of knowledge of the IEEE488 standard's controller function section.

3.4 Licenses

Q: Who are using IBF?

A: Since its introduction in 1988, IBF has been ported to many different hardware platforms. Below are some of our OEM customers (PortPak licensees):

Aero Systems Engineering, Inc., MN
Bill West Incorporated, CT
British Telecom Research Lab., England
Digalog Systems, Inc., WI
Gespac Inc., AZ
Greenspring Computers, CA
Heurikon Corporation, WI
Inducom Systems B.V., Netherlands

Lawrence Livermore National Laboratories, CA
Microboards, Inc., Japan
Microcraft, Inc., Japan
Micro-Link, IN
Microsys GmbH, Germany
Minoruta Corporation, Japan
Mizar, TX
Oettle+Reichler GmbH, Germany
Osque Systems, Japan
Rassco, Inc., Japan
Weza System Technologie GmbH, Germany
(And more)

Q: What do I need to use IBF?

A: If you are an ***end-user*** and want to use IBF on one or just a few already-existing computers, try asking your system supplier for the availability of IBF. You need to buy one copy of IBF object code license for each computer from the system supplier. If your system supplier has not ported IBF, try one of the OEM licensees listed above, or consult one of our authorized distributors listed on the last page of this booklet. Sorry, we are currently not marketing off-the-shelf versions of IBF for particular platforms.

If you are a *computer manufacturer*, *system integrator*, *value added reseller*, or end-user planning to use IBF on a quite few computers, you need to buy an IBF PortPak. IBF PortPak is distributed by our authorized distributors listed on the last page of this booklet.

Q: What is IBF PortPak?

A: An IBF PortPak is a complete starting kit for computer manufacturers, system integrators, value added resellers, and all other

parties who wish to port IBF to their hardware platforms.

An IBF PortPak includes:

- *IBF file manager program (object code)*
- *IBF example device driver (source code, assembly)*
- *IBF example device descriptor (source code, assembly)*
- *IBF utility programs (object code)*
- *IBF test utility programs (object code)*
- *IBF C language library (relocatable object code)*
- *Right to run the above IBF programs on a single computer*
- *IBF User's Manual*
- *IBF Technical Manual*
- *IBF Porting Manual.*

IBF PortPak is subject to license agreement.

Q: What is needed to sell IBF on our own computer?

A: You need to purchase an IBF object code license for each computer IBF is used on. The object code licenses are sold only to the PortPak licensees and there are significant volume purchase discounts. Call one of our authorized distributors.

An IBF object code license includes:

- *IBF file manager program (object code)*
- *IBF device driver (object code, customized)*
- *IBF example device descriptor (object code, customized)*
- *IBF utility programs (object code)*
- *IBF C language library (relocatable object code)*
- *Right to run the above IBF programs on a single computer.*

The IBF User Manual may optionally be purchased.

Q: Can I buy the whole source code?

A: Yes. We provide flexible licensing plans including the full source code license and scheduled volume purchase as well.* Call one of our authorized distributors.

3.5 Porting

Q: What LSIs does IBF support?

A: The example device driver included in an IBF PortPak supports either *TMS9914A* (TI) or *μPD7210* (NEC). There are at least two more IEEE488 LSIs commercially available today, namely 8291A (Intel) and MC68488 (Motorola). IBF, however, does not and will not support these parts because they are virtually obsolete and are known to be too buggy or cumbersome.

Q: How long is it estimated to port IBF to a brand-new hardware platform?

A: The answer may depend on several factors such as complexity of the hardware, need for DMA, the degree of experience on OS-9 porting, and so on. The example device driver source code is designed to be as flexible as possible so that only changing a few flag parameters customizes it to your own hardware. The shortest record among our customers is *only one day*, and our informal survey has shown an average of less than *three days*.

Q: What do I need to port IBF besides a PortPak?

A: The following things are necessary:

- ***Professional OS-9** or equivalent OS-9 package (V2.2 or later, from Microware)*
- ***Sysdbg** system level debugger (from Microware)*

*Subject to license agreement.

- *One or more IEEE488 devices*

We also recommend the following:*

- *The ANSI/IEEE488.1 standard book*
- *An IEEE488 bus analyzer*
- *An IEEE488 printer (e.g. HP2225A)*
- *An IEEE488 DMM (e.g. HP3478A)*

Q: What kind of IEEE488 bus analyzers are recommended for porting?

A: Any commercially available IEEE488 bus analyzer/monitor such as *HP59401A* (Hewlett-Packard), *Analyzer 488* (IO Tech), *GPIB-400* (National Instruments), and so on will be useful. For porting, however, these are really overkill; a simple bus analyzer like Ziatech's *ZT 488* with just a number of toggle switches and LEDs and simple random logic inside will be more than enough.*

A printer with an IEEE488 interface is also useful because the multiline messages on the bus can be dumped to monitor with its listen-only mode.

Q: Are there any debugging tool programs?

A: Yes. First, you can use some regular OS-9 commands such as *LIST* and *ECHO* to read and write from/to an IBF device. For IEEE488 specific functions, a set of executable command programs are included in the IBF program package.

*The devices and their manufacture names listed herein are merely for reference purposes. ARK Systems USA has no relationships with these companies, and provides no guarantee about these devices for the compatibility with IBF and about the consequential results caused by use of these devices.

3.6 Application Program Development

Q: Is there any special technique needed to develop IBF application programs?

A: No. As long as you develop your applications in C, all your existing techniques and tools will work as well. See the **Bus Management** section for the library functions.

Q: Do I need a special tool for debugging my IBF application?

A: If your application talks with more than one IEEE488 device or there is heavy bus traffic, we recommend using a good IEEE-488 bus analyzer. Refer to the previous section titled "**Porting.**"

3.7 Technical Support

Q: How do I get technical support for IBF?

A: If you are an ***end-user***, contact your system supplier first. This will screen out most hardware and implementation-specific problems. If the system supplier can't find the correct solution or the problem appears to be general, it will be relayed to us and you will get the solution from the system supplier.

If you are a *PortPak licensee*, contact the authorized IBF distributor you purchased IBF from. They are very knowledgeable about OS-9 and IBF, and will help you solve the problem.

Whenever IBF is improved or a major bug is fixed, we promptly notify you through our authorized distributors and you will receive necessary materials such as update disks and/or work-around instructions. We keep you informed.

Chapter 4
Specifications*

Product Name:

IBF — IEEE488/GP-IB Interface Bus File Manager.

CPU:

All Motorola MC68xxx CPUs supported by OS-9/68000.

OS:

OS-9/68000 V2.2 and up.

Functions:

IBF has the following entries of standard OS-9 file managers: Create, Open, Read, Write, Readln, Writeln, GetStat, SetStat, Close (MakDir, ChgDir, Delete and Seek are not supported).

Supported LSIs:

NEC μPD7210 and TI TMS9914A.

Interface Functions:

(An example implementation with the μPD7210)

SH1 . . Complete capability of source handshake interface.
AH1 . Complete capability of acceptor handshake interface.
T6 . . . . . Talker interface capability except talk only.

*Subject to change without notice.

TE0 . . . . . No capability of extended talker interface.
L4 . . . . Listener interface capability except listen only.
LE0 . . . . . No capability of extended listener interface.
SR1 . . . Complete capability of service request interface.
RL0 . . . . . No capability of remote/local interface.
PP1 Remote configuration capability of parallel poll interface.
PP2 Local configuration capability of parallel poll interface.
DC1 . . . . Complete capability of device clear interface.
DT1 . . . Complete capability of device trigger interface.
C1 . . . . . System controller interface capability.
C2 . . . . . Send IFC, controller-in-charge capabilities.
C3 . . . . . Send REN capability.
C4 . . . . . Response to SRQ capability.
C5 . . . . . Complete capability of controller.

Transfer Modes:

Binary: No end-of-line character processing (I$Read/I$Write)
Text: End-of-line character processing (I$ReadLn/I$WritLn).

DMA Transfer:

Optionally supported by the device driver.

Mutual Exclusion:

A device can be locked by a process.

Conforming Standards:

ANSI/IEEE488.1-1987, JIS C1901, IEC 625-1.

C Library Functions:

_ib_abort(), _ib_bus_status(), _ib_card_ppoll_resp(), _ib_clear(), _ib_lmsg(), _ib_local(), _ib_local_lockout(), _ib_loack(), _ib_pass_ctl(), _ib_ppoll(), _ib_ppoll_config(), _ib_ppoll_resp_ctl(), _ib_ppoll_unconfig(), _ib_remote(), _ib_ren_ctl(), _ib_rqst_srvce(), _ib_send_cmnd(), _ib_set_sig(), _ib_spoll(), _ib_status_wait(), _ib_timeout_ctl(), _ib_trigger(), _ib_unlock()

Appendix
Authorized Distributors

IBF is distributed and supported by the following authorized distributors; please inquire prices, terms, and conditions of:

■AMERICA:

Windsor Systems
2407 Lime Kiln Lane, Louisville, KY 40222 U.S.A.
Phone: 502-425-9560/Fax: 502-426-3944

■ASIA:

Microboards, Inc.
Takase-Cho 31-8, Funabashi-Shi, Chiba 273 Japan
Phone: 0474-37-9808/Fax: 0474-37-9822

Osque Systems, Inc.
Hoei Plaza #902, Ogikubo 5-30-17, Suginami-Ku, Tokyo 167 Japan
Phone: 03-3220-0384/Fax: 03-3220-0417

■EUROPE:

Galactic Industrial, Ltd.
Unit 3B, Mountjoy Research Centre, Stockton Road,
Durham, DH1 3UR England
Phone: 091-3848343/Fax: 091-3847742

Snowtop Computers, Ltd.
45 Summer Street, Slip End, Bedfordshire, LU1 4BL England
Phone: 0582-451084/Fax: 0582-20764